Algorithms

ALGORITHM 30
NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION

K. W. Ellenberger
Missile Division, North American Aviation, Downey, California

```
procedure  ROOTPOL (n, a, L, F, u, v, CONV)  ;
           value n, a, L, F  ; integer L, F, n  ;
           array a, u, v, CONV  ;
comment The Bairstow and Newton correction formulae are
  used for a simultaneous linear and quadratic iterated synthetic
  division. The coefficients of a polynomial of degree n are given as
  a_i (i = 0, 1, ... , n) where a_n is the constant term. The coeffi-
  cients are scaled by dividing them by their geometric mean.
  The Bairstow or Newton iteration method will nearly always
  converge to the number of figures carried, F, either to root
  values or to their reciprocals. If the simultaneous Newton and
  Bairstow iteration fails to converge on root values or their
  reciprocals in L iterations, the convergence requirement will be
  successively reduced by one decimal figure. This program antici-
  pates and protects against loss of significance in the quadratic
  synthetic division. (Refer to "On Programming the Numerical
  Solution of Polynomial Equations," by K. W. Ellenberger,
  Commun. ACM 3 (Dec. 1960), 644-647.) The real and imaginary
  part of each root is stated as u[i] and v[i], respectively, together
  with the corresponding constant, CONV_i, used in the con-
  vergence test. This program has been used successfully for over
  a year on the Bendix G15-D (Intercard System) and has recently
  been coded for the IBM 709 (Fortran System);
            begin integer i, j, m  ; array h, b, c, d, e[-2 :n]  ;
              real t, K, ps, qs, pt, qt, s, rev, r  ;
ROOTPOL:   b_-1 := b_-2 := c_-1 := c_-2 := d_-1 := d_-2 := e_-1 :=
              e_-2 := 0  ;
           for j := 0 step 1 until n do h_j := a_j  ;  t := 1  ;
              K := 10^F  ;
ZROTEST:   if h_n = 0 then
           begin u_n := 0  ;  v_n := 0  ;  CONV_n := K  ;
              n := n - 1  ;  go to ZROTEST
           end  ;
INIT:      if n = 0 then go to RETURN  ;
              ps := qs := pt := qt := s := 0  ;
              rev := 1  ;  K := 10^F  ;
           if n = 1 then
           begin r := - h_1/h_0  ;  go to LINEAR
           end  ;
           for j := 0 step 1 until n do
           begin
           if h_j = 0 then s := s else s := s + log(abs(hj))
           end  ;  s := s^10  ;
           for j := 0 step 1 until n do h_j := h_j/s  ;
           if abs (h_1/h_0) < abs (h_n-1/h_n) then
REVERSE:   begin t := -t  ;  m := entier ((n+1)/2)  ;
           for j := 0 step 1 until m do
           begin s := h_j  ;  h_j := h_n-j  ;  j_n-j := s
           end
           end  ;
           if qs ≠ 0 then
           begin p := ps  ;  q := qs  ;  go to ITERATE
           end  ;
           if h_n-2 = 0 then
           begin q := 1  ;  p := -2
           end else
           begin q := h/h_n-2  ;  p := (h_n-1 - q × h_n-3)/h_n-2
           end  ;
           if n = 2 then go to QADRTIC  ;  r := 0  ;
ITERATE:   for i := 1 step 1 until L do
           begin
BAIRSTOW:  for j := 0 step 1 until n do
           begin b_j := h_j - p × b_j-1 - q × b_j-2  ;
             c_j := b_j-p × c_j-1 - q × c_j-2
           end  ;
           if n_n-1 = 0 then go to BNTEST  ;
           if b_n-1 = 0 then go to BNTEST  ;
           if abs (h_n-1/b_n-1)<K then go NEWTON  ;
             b_n := h_n-q × b_n-2  ;
BNTEST:    if b_n = 0 then go to QADRTIC  ;
           if K < abs (h_n/b_n) then go to QADRTIC  ;
NEWTON:    for j := 0 step 1 until n do
           begin d_j := h_j + r × d_j-1  ;  e_j := d_j + r × e_j-1
           end  ;
           if d_n = 0 then go to LINEAR  ;
           if K < abs (h_n/d_n) then go to LINEAR  ;
             c_n-1 := -p × c_n-2-q × c_n-3  ;
             s := c^2_n-2 - c_n-1 × c_n-3  ;
             if s=0 then
           begin p := p - 2  ;  q := q × (q + 1)
           end else
           begin p := p + (b_n-1 × c_n-2 - b_n × c_n-3)/s  ;
             q := q + (-b_n-1 × c_n-1 + b_n × c_n-2)/s
           end  ;
           if e_n-1 = 0 then r := r-1 else r := r - d_n/e_n-1
           end  ;  ps := pt  ;  qs := qt  ;  pt := p  ;
             qt := q  ;
           if rev < 0 then K := K/10  ;  rev = -rev  ;
             go to REVERSE  ;
LINEAR:    if t < 0 then r := 1/r  ;  u_n := r  ;  v_n := 0  ;
           CONV_n := K  ;  n := n-1  ;
           for j := 0 step 1 until n do h_j := d_j  ;
           if n = 0 then go to RETURN  ;
             go to BAIRSTOW  ;
QADRTIC:   if t < 0 then
           begin p := p/q  ;  q := 1/q
           end  ;
           if 0 < (q - (p/2)^2) then
           begin u_n := u_n-1 := -p/2  ;
             s := sqrt (q - (p/2)^2)  ;  v_n := s  ;
             v_n-1 := -s
           end else
           begin s := sqt ((p/2)^2) - q)  ;
           if p < 0 then u_n := -p/2 + s
             else u_n := -p/2-s  ;  u_n-1 := q/u_n  ;
             v_n := v_n-1 := 0
           end  ;  CONV_n := CONV_n-1 := K  ;
             n := n-2  ;
             for j := 0 step 2 until n do h_j := b_j  ;
             go to INIT  ;
RETURN:    end
```

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May, 1960). Contributions should be sent to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D.C. Algorithms should be in the Publication form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material and no responsibility is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.